MSA

**DEFINITION**

Microservices is an architecture style in which a single application is composed of many micro or small services.

These small services are loosely coupled and independently deployable

The major advantage we have using microservice is that even if one component fails, the entire software system will not break.

The small services

1. runs in its **own process** and
2. **Communicating with lightweight mechanisms** eg: http
3. **Independently deployable**

**KEY ENABLERS FOR MSA**

(i) Availability of containers, container orchestration, and management platforms

(ii) Availability of communication protocols

(iii) Availability of simplified API design mechanisms

(v) Availability of design patterns, service repositories, and tools

**ADVANTAGES**

* Support for a decentralized approach for software development
* MSA provides loosely coupled and individual deployment
* frequent deployment, and continuous delivery
* High scalability at individual service level and at low cost
* Support of code reuse
* Better fault isolation

**LIMITATIONS OF MSA**

Complexity in managing a large number of services—In MSA follows Single Responsibility Principle (SRP) which causes the application to consist of several services.

Multiple databases make management and transaction difficult—In MSA, each microservice should have its own related tables or databases.

Testing: Integration testing, as well as end-to-end testing, can become more difficult, and more important than ever. (Testing can be little time consuming as each independent service needs to be tested before integrated testing.

Versioning: When you update to new versions, keep in mind that you might break backward compatibility.

Deployment: To make deployment easier, you must first invest in quite a lot of automation as the complexity of microservices becomes overwhelming for human deployment.

Logging: With distributed systems, you need centralized logs to bring everything together.

Monitoring: It’s critical to have a centralized view of the system to pinpoint sources of problems.

Debugging: Remote debugging through your local integrated development environment (IDE) isn’t an option and it won’t work across dozens or hundreds of services

**MONOLITHIC**

A Monolithic application is built as one single unit in which the user interface and data access code are combined into a single program on a single platform.

Enterprise Monolithic applications are built in three parts:

A database, consisting of many tables usually in a relational database management system.

Client-side user interface consisting of HTML and/or JavaScript running in a browser.

Server-side applications which work as the middle man between the user interface and the database will handle HTTP requests

**ADVANTAGES**

**Easy development**

**Easy deployment**

**Performance** – In a centralized code base and repository, one API can often perform the same function that numerous APIs perform with microservices.

**Simplified testing** – Since a monolithic application is a single, centralized unit, end-to-end testing can be performed faster than with a distributed application.   
   
**Easy debugging** – With all code located in one place, it’s easier to follow a request and find an issue.

**LIMITATIONS OF MONOLITHIC**

* The modules of monolithic are tightly coupled. They are Designed, developed, and deployed (DDD) as a single unit or a single executable jar file.
* It’s hard to practice Agile development and delivery methodologies. (CI/CD)
* Frequent deployment becomes very tough
* You must redeploy the entire application in order to update any part of it
* One unstable service can bring the whole application down
* Difficult to adopt new technologies
* Many development teams will be working on the same project. They are not actually very independent

|  |  |
| --- | --- |
| Monolithic | Microservices |
| Single service/application **should contain all the business functionality** | Single service should contains only one business functionality |
| All service are **tightly coupled** | All services are loosely coupled |
| Application is developed in **one single programming language** | Each service **can be** in **different programming language** |
| **Single database** for all services. | Each service **has separate database** |
| **All services run in same process so if one service goes down then whole application breaks** | Each service runs in different process so failure of one service does not affects other services |
| **Difficult to scale** | Can be Scaled easily |
| **Single large team works on whole application** | Separate small team work on each Service which are more focused. |

**SERVICE ORIENTED ARCHITECTURE SOA**

Service-oriented architecture is an architectural style which breaks monolithic applications into a series of smaller services.

Service-orientation is a way of thinking in terms of services

SOA allows users to combine a large number of facilities from existing services to form applications.

Each service in SOA is a complete business function in itself.

The services are published in such a way that it makes it easy for the developers to assemble their apps using those services

A service has four properties:

* It logically represents a repeatable business activity with a specified outcome. (e.g., check customer credit, provide weather data).
* It is self-contained.
* It is a black box for its consumers, (meaning the consumer does not have to be aware of the service's inner workings).
* It may be composed of other services

An important aspect of SOA is the separation of the service interface (the what) from its implementation (the how)

The consumers are only concerned about the service interface and do not care about its implementation.

SOA-based systems can therefore function independently of development technologies and platforms (such as Java, .NET, etc.)

**ADVANTAGES**

**What are the similarities between SOA and MSA**

At first glance, the two approaches ie SOA and MSA sound very similar, and in some ways, they are**.**

1. Both SOA and MSA are service-based architectures
2. They share platform independence, language independence, and location independence
3. **Both involve cloud or** [hybrid cloud](https://www.ibm.com/cloud/learn/hybrid-cloud) **environments for agile application development** and **deployment**, and
4. **both can scale to meet the speed and operational demands of big data**.
5. Both break large, complex applications into **small, flexible components that are easier to work with**.
6. And both differ from a **traditional, monolithic architecture in that every service has its own responsibility.**

**Microservices vs. SOA: What’s the Difference?**

# 

|  |  |
| --- | --- |
| **Microservice Based Architecture (MSA)** | **Service-Oriented Architecture (SOA)** |
| They are built **to perform a single business task.** | SOA applications are built to perform **numerous business tasks.** |
| Each microservice have an **independent** database. | SOA services share the **whole** data storage. |
| It focuses on **decoupling**. | It focuses on application service **reusability**. |
| It uses a **simple messaging system** for communication. Microservices uses **lightweight protocols** such as **REST**, and **HTTP**, etc. | It uses **Enterprise Service Bus** (ESB) for communication. |
| Microservices follows "**share as little as possible**" architecture approach. | SOA follows "**share as much as possible architecture**" approach. |
| Microservices are much better in **fault tolerance** in comparison to SOA. | SOA is not better in fault tolerance in comparison to MSA. |

|  |  |
| --- | --- |
| **Spring** | **SpringBoot** |
| Spring is an open-source lightweight framework widely used to develop enterprise applications. | Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs. |
| The most important feature of the Spring Framework is dependency injection. | The most important feature of the Spring Boot is Autoconfiguration. |
| It helps to create a loosely coupled application. | It helps to create a stand-alone application. |
| To run the Spring application, we need to set the server explicitly. | Spring Boot provides embedded servers such as Tomcat and Jetty etc |
| To create a Spring application, the developers write lots of code. | It reduces the lines of code. |
| It doesn’t provide support for the in-memory database. | It provides support for the in-memory database such as H2. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Microservices | SOA | Monolithic |
| Design | Services are built in small units and expressed formally with business-oriented APIs. | Services can range in size anywhere from small application services to very large enterprise | Monolithic applications evolve into huge size, a situation where understanding application is difficult. |
| Usability | Services exposed with a standard protocol, such as a RESTful API, and consumed/reused by other services and applications. | Services exposed with a standard protocol, such as SOAP and consumed/reused by other services – | Limited re-use is realized across monolithic applications. |
| Scalability | Services exist as independent deployment artifacts and can be scaled independently of other services. | Dependencies between services and reusable sub-components can introduce scaling challenges. | Scaling monolithic applications can often be a challenge. |
| Agility | Smaller independent deployable units ease build/release management, thereby high operational agility  . | Enhances components sharing that increases dependencies and limits management capabilities. | Difficult to achieve operational agility in the repeated deployment of monolithic application artifacts. |
| Development | Developing services discretely allows developers to use the appropriate development framework for the task at hand. | Reusable components and standard practices helps developers with implementation. | Monolithic applications are implemented using a single development stack (i.e., JEE or .NET), which can limit the availability of “the right tool for the job”. |